1. Tempo disponibile 120 minuti (90 minuti per gli studenti di "Introduzione agli Algoritmi" - 6 CFU, che devono fare solo i primi 3 esercizi).
2. Non è possibile consultare appunti, slide, libri, persone, siti web, ecc.
3. Scrivere in modo leggibile, su ogni foglio, nome, cognome e numero di matricola.
4. Le soluzioni agli esercizi che richiedono di progettare un algoritmo devono:
   - spiegare a parole l'algoritmo (se utile, anche con l'aiuto di esempi o disegni),
   - fornire e commentare lo pseudo-codice (indicando il significato delle variabili),
   - calcolare la complessità (con tutti i passaggi matematici necessari),
   - se l'esercizio ammette più soluzioni, a soluzioni computazionalmente più efficienti e/o concettualmente più semplici sono assegnati punteggi maggiori.

1. Calcolare la complessità $T(n)$ del seguente algoritmo MYSTERY:

Algorithm 1: MYSTERY(INT n) $\rightarrow$ INT

```
if m ≤ 1 then
|  return 1
else
|  i ← 1
|  j ← 0
|  while i ≤ n do
|  |  i ← i + 2
|  |  j ← j + MYSTERY2(n/2)
|  end
|  return j + MYSTERY(n/4) + MYSTERY(n/4) + MYSTERY(n/4)
end

function MYSTERY2(INT m) → INT
if m ≤ 1 then
|  return 1
else
|  j ← 1
|  while j ≤ m do
|  |  j ← j + 1
|  end
|  return j + MYSTERY2(m/3) + MYSTERY2(m/3)
end
```

Soluzione. Iniziamo dall'analisi del costo computazionale $T'(m)$ della funzione MYSTERY2. Si tratta di una funzione che soddisfa la seguente relazione di ricorrenza:

$$T'(m) = \begin{cases} 1 & \text{se } m = 1 \\ 2 \times T'(\frac{m}{3}) + m & \text{altrimenti} \end{cases}$$

in quanto la funzione è ricorsiva con costo costante nel caso base e, nel caso ricorsivo, con due chiamate ricorsive in cui si riduce di 3 volte il parametro più un ciclo di costo lineare. Tale relazione di ricorrenza può essere risolta utilizzato il Master Theorem considerando i parametri $a = 2$, $b = 3$ e $\beta = 1$ che implicano $\alpha = \frac{\log 2}{\log 3} = \frac{1}{\log 3}$. Avendo $\alpha < \beta$ ci troviamo nel caso 3 del teorema che implica $T'(m) = \Theta(m)$.

Passiamo ora all'analisi del costo computazionale $T(n)$ della funzione MYSTERY. Si tratta di una funzione che soddisfa la seguente relazione di ricorrenza:

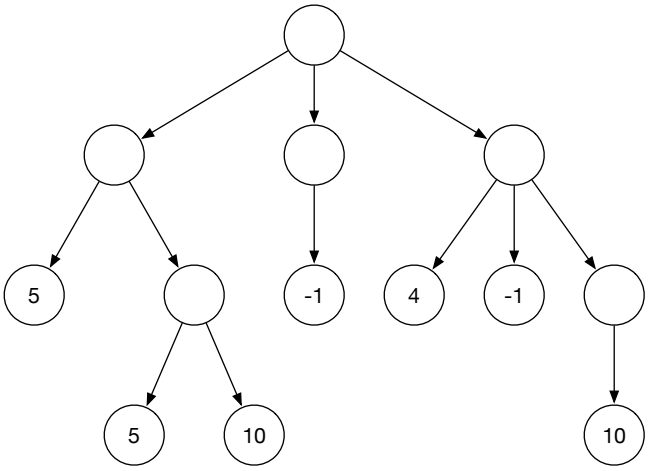
$$T(n) = \begin{cases} 1 & \text{se } m = 1 \\ 3 \times T(\frac{n}{4}) + \frac{n}{2} \times T'(\frac{n}{2}) & \text{altrimenti} \end{cases}$$

in quanto la funzione è ricorsiva con costo costante nel caso base e, nel caso ricorsivo, con tre chiamate ricorsive in cui si riduce di 4 volte il parametro più un ciclo eseguito $n/2$ volte (in quanto ad ogni loop si incrementa di 2 l'indice di controllo del ciclo) che contiene una chiamate alla funzione MYSTERY2 con parametro $n/2$. Abbiamo che $T'(\frac{n}{2}) = \Theta(\frac{n}{2})$ e quindi $\frac{n}{2} \times T'(\frac{n}{2}) = \Theta(n^2)$. La relazione di ricorrenza può quindi essere riscritta come segue:

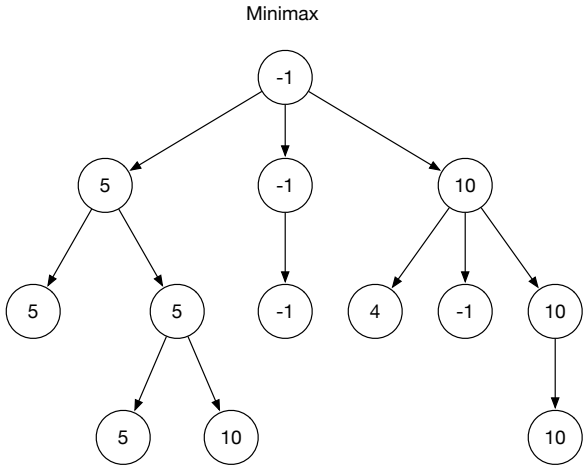
$$T(n) = \begin{cases} 1 & \text{se } m = 1 \\ 3 \times T(\frac{n}{4}) + \Theta(n^2) & \text{altrimenti} \end{cases}$$

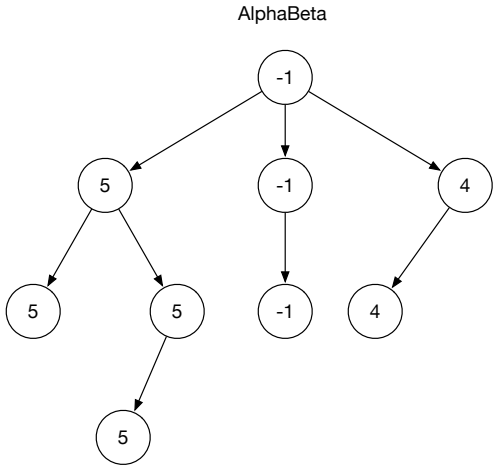
Tale relazione di ricorrenza può essere risolta utilizzato il Master Theorem considerando i parametri $a = 3$, $b = 4$ e $\beta = 2$ che implicano $\alpha = \frac{\log 3}{\log 4}$. Avendo $\alpha < \beta$ ci troviamo nel caso 3 del teorema che implica $T(n) = \Theta(n^2)$.

2. Dato il seguente albero, con label numeriche assegnate alle foglie, mostrare le label dei nodi intermedi dopo l'applicazione degli algoritmi Minimax e AlphaBeta pruning assumendo che la radice dell'albero sia un nodo di **minimizzazione** (i.e. prende il valore minimo tra i valori assegnati ai figli diretti). Nel disegnare l'albero dopo l'applicazione dell'AlphaBeta pruning non mostrare i rami dell'albero non visitati.



Soluzione. Gli alberi risultanti dopo l'applicazione dell'algoritmo Minimax e AlphaBeta pruning sono mostrati di seguito.





Cont.

3. Su un pallone aerostatico in difficoltà di volo si deve ridurre il peso complessivo di un valore W (numero naturale), buttando fuori dall'abitacolo alcuni oggetti. Esistono n oggetti che si possono buttare, ognuno con un proprio peso (un numero naturale) ed un proprio valore (un numero non negativo). Si vuole trovare la combinazione di oggetti ottimale, ovvero un insieme di oggetti di peso complessivo superiore a W che minimizzi il valore complessivo. Progettare un algoritmo che dati i vettori $P[1..n]$ e $V[1..n]$ che rappresentano i pesi ed i valori degli n oggetti, ed il valore W di peso da ridurre, stampa un insieme ottimale di oggetti da buttare dal pallone aerostatico per ridurre il peso di almeno una quantità W, minimizzando il valore complessivo che viene gettato.

   Soluzione. È possibile utilizzare la programmazione dinamica considerando i problemi $T(i,j)$, con $i \in \{1 \dots n\}$ e $j \in \{1 \dots W\}$, tale che $T(i,j)$ = valore complessivo minimo di un insieme di oggetti nell'insieme $\{1 \dots i\}$ con peso complessivo maggiore o uguale a j (∞ nel caso in cui non esista un tale insieme). I problemi $T(i,j)$ possono essere risolti considerando che:

$$T(i,j) = \begin{cases} \infty & \text{se } i = 1 \text{ e } P[1] < j \\ V[1] & \text{se } i = 1 \text{ e } P[1] \geq j \\ min\{V[i], T(i-1,j)\} & \text{se } i > 1 \text{ e } P[i] \geq j \\ min\{V[i] + T(i-1, j-P[i]), T(i-1,j)\} & \text{se } i > 1 \text{ e } P[i] < j \end{cases}$$

   L'Algoritmo 2 risolve i problemi $T(i,j)$ inserendo le relative soluzioni nella tabella $T[1..n, 1..W]$. L'esercizio richiede di stampare l'insieme di oggetti ottimale (e non il loro valore memorizzato in $T[n,W]$); a tal fine l'algoritmo utilizza una matrice di booleani $B[i,j]$ che indica se l'oggetto i-esimo fa parte della soluzione al problema $T(i,j)$. Abbiamo che il costo computazionale di tale algoritmo risulta essere $T(n,W) = \Theta(n \times W)$ in quanto le operazioni elementari hanno tutte costo costante, ed il blocco di maggiore costo è quello che contiene i due **for** annidati le cui operazioni interne vengono eseguite $(n-1) \times W$ volte.

Cont.

Algorithm 2: PALLONE(INT $P[1..n]$, INT $V[1..n]$, INT W)

```
// Inizializzazione prima riga delle tabelle T e B
for j ← 1 to W do
    if P[1] < j then
        T[1, j] ← ∞; B[1, j] ← false
    else
        T[1, j] ← V[1]; B[1, j] ← true
    end
end
// Riempimento restanti righe delle tabelle T e B
for i ← 2 to n do
    for j ← 1 to W do
        if P[i] ≥ j and V[i] < T[i − 1, j] then
            T[i, j] ← V[i]; B[i, j] ← true                    // si prende solo l'oggetto i-esimo
        else if P[i] < j and V[i] + T[i − 1, j − P[i]] < T[i − 1, j] then
            T[i, j] ← V[i] + T[i − 1, j − P[i]]; B[i, j] ← true    // si prende anche l'oggetto i-esimo
        else
            T[i, j] ← T[i − 1, j]; B[i, j] ← false              // non si prende l'oggetto i-esimo
        end
    end
end
if T[n, W] ≠ ∞ then
    // Esiste una soluzione e viene stampata
    pesoRimanente ← W; oggetto ← n
    while pesoRimanente > 0 do
        if B[oggetto, pesoRimanente] then
            stampa(oggetto)
            pesoRimanente ← pesoRimanente − P[oggetto]
        end
        oggetto ← oggetto − 1
    end
end
```

4. Una rete autostradale viene rappresentata tramite un grafo non orientato pesato in cui i vertici rappresentano città, gli archi rappresentano tratti autostradali di collegamento fra città, ed i pesi degli archi rappresentano i costi dei relativi pedaggi autostradali. Inoltre, ogni volta che si attraversa una città, è necessario pagare una tassa di attraversamento (oltre ai costi dei pedaggi autostradali dei tratti usati per raggiungere e per uscire dalla città). Quindi, dato un cammino $v_0, v_1, \ldots v_k$, il costo complessivo di tale cammino è dato dalla somma di tutti i pedaggi per i tratti (v_i, v_{i+1}), con $i \in \{0 \ldots k-1\}$ più le tasse di attraversamento delle città v_j, con $j \in \{1 \ldots k-1\}$. Progettare un algoritmo che dato un grafo non orientato pesato $G = (V, E, w)$ con pesi non negativi, una funzione di costo $t : V \to \mathbb{R}$ tale che $\forall v \in V \,.\, t(v) \geq 0$, un nodo sorgente $s \in V$ ed uno destinazione $d \in V$, stampa un cammino di costo complessivo minimo per andare da s a d.

   Soluzione. Per cercare il cammino minimo fra due vertici dati è possibile utilizzare una versione modificata dell'algoritmo di Dijkstra in quanto i costi degli archi, e dei pedaggi, risultano essere non negativi. La modifica rispetto all'algoritmo tradizionale deriva dal fatto che, nel momento del calcolo della distanza di un vertice v, si deve prendere in considerazione anche il costo $t(v)$ del pedaggio per entrare nella relativa città.

   L'Algoritmo 3 utilizza la solita convenzione secondo cui i vertici vengono rappresentati da numeri interi nell'intervallo $\{1 \ldots |V|\}$; in questo modo la funzione $t : V \to \mathbb{R}$ viene rappresentata tramite un array di $|V|$ numeri (un numero di costo del pedaggio per ogni vertice). L'algoritmo ha il medesimo costo computazionale dell'algoritmo di Dijkstra, quindi $T(n, m) = O(m \log n)$ con $n = |V|$ e $m = |E|$.

Cont.

Algorithm 3: CamminoPiuEconomico(Grafo $G = (V, E, w)$, Number $t[1..|V|]$, Int s, Int d)

```
// inizializzazione strutture dati
n ← G.numNodi()
Int pred[1..n], v, u
Number D[1..n]
for i ← 1 to n do
    D[i] ← ∞
    pred[i] ← −1
end
D[s] ← 0
MinPriorityQueue[Int, Number] Q ← new MinPriorityQueue[Int, Number]()
Q.insert(s, D[s])

// esecuzione algoritmo di Dijkstra (modificato)
while not Q.isEmpty() do
    u ← Q.findMin()
    Q.deleteMin()
    for v ∈ u.adjacent() do
        if D[v] = ∞ then
            // prima volta che si incontra v
            D[v] ← D[u] + w(u, v) + t[v]
            Q.insert(v, D[v])
            pred[v] = u
        else if D[u] + w(u, v) + t[v] < D[v] then
            // scoperta di un cammino migliore per raggiungere v
            Q.decreaseKey(v, D[v] − D[u] − w(u, v) − t[v])
            D[v] = D[u] + w(u, v) + t[v]
            pred[v] = u
        end
    end
end
// stampa del cammino da s a d
printPath(pred, d)

// funzione ausiliaria di stampa del cammino
procedure printPath(Int pred[1..|V|], Int d)
if pred[d] ≠ −1 then
    printPath(pred, pred[d])
    print d
end
```

The End.